A Component-Based, Distributed Object Services Architecture for a Clinical Workstation

Henry C. Chueh M.D. M.S., Wayne F. Raila, John J. Pappas, Mark Ford,
Polina Zatsman, John Tu M.D., G. Octo Barnett, M.D.
Laboratory of Computer Science, Massachusetts General Hospital

Attention to an architectural framework in the development of clinical applications can promote reusability of both legacy systems as well as newly designed software. We describe one approach to an architecture for a clinical workstation application which is based on a critical middle tier of distributed object-oriented services. This tier of network-based services provides flexibility in the creation of both the user interface and the database tiers. We developed a clinical workstation for ambulatory care using this architecture, defining a number of core services including those for vocabulary, patient index, documents, charting, security, and encounter management. These services can be implemented through proprietary or more standard distributed object interfaces such as CORBA and OLE. Services are accessed over the network by a collection of user interface components which can be mixed and matched to form a variety of interface styles. These services have also been reused with several applications based on World Wide Web browser interfaces.

INTRODUCTION

Many clinical applications will ultimately reuse similar application logic in order to perform a common task. An example is the need to perform a "patient lookup" in virtually every clinical application. Traditional approaches to solving this problem range from using a database-centric model where each application uses its own program code to access the same central database, to application-centric models where modular code, usually in the form of subroutines, is reused in every application. The advent of client-server architectures which divide an application into at least two potential parts have highlighted the issue of where to locate application logic - on the client or on the server.[1] There has been recent focus on insulating database implementations from client applications. This approach can be effective for easing the migration from legacy systems.[2] Insulation of content can occur when data from the database is translated into a common clinical content format before presentation to the client.[3] Insulation can also be promoted through a technical infrastructure of a common software "bus" providing application services.[4,5] Like a hardware bus, a software bus provides a standard interface to lower-level functionality. Frameworks promoting horizontal integration across applications through a layered, component-based approach have also been described.[5,6,7] A component is a reusable software module which has two critical features: 1) it has an explicit interface which hides the complexity of the software code within, and 2) it is built using a standard software architecture which allows components to interact with each other, even if the programming languages used within two components are different. Architectures which promote this interoperability include emerging standards such as the Object Management Group's Common Object Request Broker Architecture (CORBA) and Microsoft's Object Linking and Embedding (OLE) specifications. New technologies and tools facilitate a services-based approach which allows an application to be built in multiple tiers: an explicit user interface, a middle tier of network-based application services, and a database tier. We have taken an approach which combines all of these concepts -- common data formats, distributed services, component-based applications -- and implemented an architecture to build an outpatient clinical workstation for patient records.

METHODS

Scenarios of outpatient encounters were described in writing and used as the basis for identifying potential objects and processes. This technique has been described in many object-oriented analysis methods.[8] Objects which represent core clinical entities such as *patient*, *practitioner*, and *encounter* were identified first. These entities have been identified in many other representations of clinical domains, and potential standards were reviewed for comparison and generation of a full set of object attributes.[9] Relationships between these clinical entities were established; for example, "one patient can have many encounters", or "one or more practitioners participates in an encounter".

Additional objects were defined to allow for clinician observations to be recorded as part of the electronic patient record. Comprehensive efforts to model clinical observations and concepts for the medical record have not yet led to a clear consensus on how such entities should be modeled.[3,10] However, there is an increasing emphasis towards structured records, which we also favor. We focused on defining objects which correspond to the typical recording habits of practitioners. For example, visit notes often have loose narrative sections such as "Reason for visit", or "Plans", leading to the definition of a *Narrative* object class, while a more structured entity such as

0195—4210/96/$5.00

clinical problem is identified as a separate *Problem* class. Next, a collection of coordinating or "manager" objects were defined as objects which would provide services to manipulate the clinical data objects. These manager objects are derived from a need to model functional processes rather than from a need to represent clinical data. For instance, the PatientIndexManager object provides a method to perform a patient lookup and create a *Patient* object, and a DocumentManager object provides a method to retrieve a list of *Document* objects for a given *Patient*.

Legacy systems and databases were reviewed for their data content. A large clinical results repository is maintained on a Tandem mainframe computer at the Massachusetts General Hospital (MGH), the site of this development. Identifying which objects needed to be constructed from this database guided the development of services such as the TestManager and the PatientIndexManager.

An application framework was defined which supports a component-based approach to constructing the user interface displays. Display components were then selected for construction and created to utilize the core applications services and data objects. We chose to implement the architecture and application using a distributed object-oriented development environment from Forté Software. This general type of environment has been well-described.[11] Network-based services created with this development environment can be brokered through proprietary as well as evolving standard object interfaces such as CORBA and OLE. The workstations are Pentium-based PCs, and middle tier servers are a combination of Unix and Windows NT servers. Underlying databases are Tandem's Non-Stop SQL and Oracle (Figure 1).

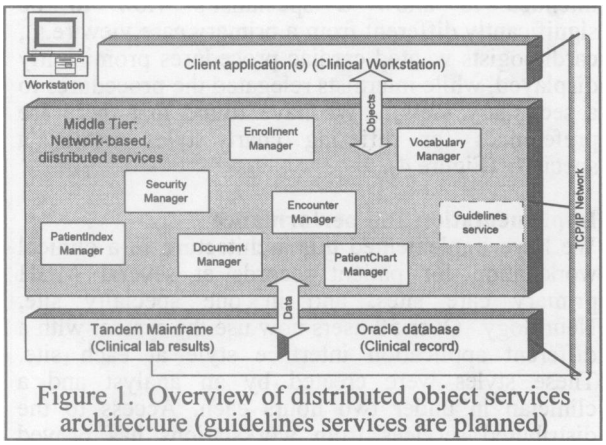

Figure 1. Overview of distributed object services architecture (guidelines services are planned)

RESULTS

Application of our methods resulted in a clinical workstation for ambulatory patient records which utilizes a collection of distributed, network-based object services as core application services. These major services enforce application logic and mediate between the database and the client. Some core services and our component-based approach towards the utilization of these services are described here.

Objects and services

We found a clear distinction in our architecture between "data objects" and objects destined to become services ("service objects"). Data objects such as *Problem* (subclass of *Observation*) consist of mostly data attributes, have few methods, and store actual patient data. Figure 2 describes briefly the attributes of the *Problem* class; attributes above the line are common to all *Observations*, those below are specific to the *Problem* subclass. Service objects such as EncounterManager consist primarily of methods, have few attributes, and do not maintain any patient data or state. The data objects we defined are part of a shallow inheritance hierarchy, while the service objects do not participate in any inheritance. In general, service objects act on and manage data objects, and coordinate processes with other services. These services can enforce business policies at a broad level which is beyond the scope of any single data object. A number of core services for patient records were defined (Table 1).

Problem *: Observation*	
dateTimeNoted	(when observed)
author	(who is observing)
concept	(controlled vocabulary code)
patient	(patient identification)
sensitivity	(level of confidentiality)
draftStatus	(draft, preliminary, or final)
thread	(links to previous *Observ.*)
dateOfOnset	(when problem started)
dateOfResolution	(when problem is resolved)
comment	(descriptive text/note)

Figure 2. *Problem* class (subclass of *Observation*)

Data objects	**Service objects**
Person	PatientIndexManager
Patient	EncounterManager
Practitioner	PatientChartManager
Encounter	DocumentManager
Observation	EnrollmentManager
Narrative	TestManager
Problem	SecurityManager
Medication	SessionManager
Allergy	VocabularyManager
Test	
Document	
Problem list	

Table 1. A selection of some major data objects and services. Subclasses are indented.

Document and chart services
The DocumentManager and PatientChartManager services form the centerpiece of structured data entry and data views. Practitioners enter data through the creation of visit documents. The document object allows a flexible mix of structured and unstructured data. This is achieved by a document which has multiple sections, some of which are more structured (e.g., Problems) than others (e.g., Narrative History). While the client application must know the definition of the *Document* class in order to display document objects to the clinical user, it does not need to understand details of how to retrieve or save them. The DocumentManager has methods to retrieve existing documents, and to create and save new ones.

The PatientChartManager provides summary views of the patient chart. For example, *GetProblemList()* and *GetAllergyList()* are two typical methods. While the DocumentManager provides the traditional encounter-based view of chart documents, this service summarizes data *across* visits and documents. This allows clinical data entered as documents to be filtered into many alternative views, and help eliminate redundant data entry. For example, when the clinician saves a document which contains new problems, these problems are retrieved automatically by the PatientChartManager as part of the problem list. In this way, the problem list is automatically generated from the entry of visit notes (Figure 3).

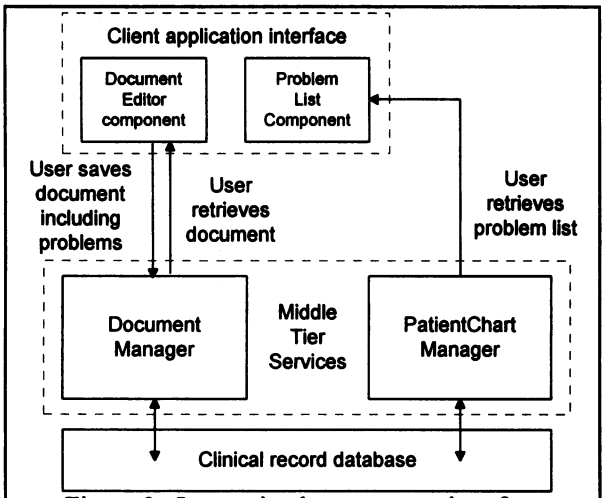

Figure 3. Interaction between user interface components and services

Vocabulary services
Distributed vocabulary servers and services have been described both theoretically and practically.[12,13] We also defined a specific vocabulary service which provides a number of critical functions: 1) Real-time lookup of concepts – user enters a phrase and a list of potential matches are retrieved from the vocabulary server, 2) Coding of structured information entered as part of a document, and 3) translation of coded information among different nomenclatures using the UMLS Metathesaurus. By incorporating a vocabulary concept as an attribute of almost all data objects, services can utilize this coded information to enforce specific application behavior. For example, when the same clinical Problem is noted as part of many visit notes for one patient, but even by different authors, the PatientChartManager keeps these problems together in a single thread, much like discussion threading in e-mail discussion groups.

Security service
The security service's central role is the authorization for use of all middle layer services. We have adopted a Kerberos approach. Kerberos is a security model developed to address network-based client server computing. It requires the user to authenticate once and obtain a ticket, and use that ticket to authorize with a security service every time an application service is used. We defined user roles with specific privileges. For example, a medical assistant can view the problem list, but cannot create a new document.

Display components
The user interface was developed in a component-based fashion. Each component utilizes middle tier services to retrieve and save the data needed for display to the user. Over a dozen component types have been created so far, and include a many list viewers (e.g., problem, allergy, medication, procedure, past medical history, encounter lists), several lab test displays, and a document view. Each of these components is designed to work independently within the application framework, allowing us to build an interface editor which allows an analyst to mix and match components to design an interface style without programming. These styles are maintained in the database, and each user gets a specific style when logging on to the application. The flexibility of this model allows us to rapidly define different visual placements of the same patient chart data for different clinical practices. This was intended to allow a specialist's view to be significantly different from a primary care view (e.g., cardiologists wanted cardiac procedures prominently displayed, while internists relegated the procedures to a secondary view). We have found that there are preferences for differing chart styles within a specialty (Figure 4).

Implementation and performance
We have implemented this architecture in a clinical workstation for patient records at several MGH primary care sites, and in one specialty site, Neurology. Over 50 users now use the system with a different application interface style at each site. These styles were created by an analyst and a clinician in under two hours each. Access to the distributed services from workstations has proved robust, with a stable network being a minimum requirement. Performance of this architecture which has an additional layer of abstraction is good. Typical screens with several display components are shown in 1-2 seconds. Access of clinical results through the middle layer TestManager service to the Tandem mainframe clinical repository also performs well with response times which rival and even surpass the native terminal interface to the Tandem.

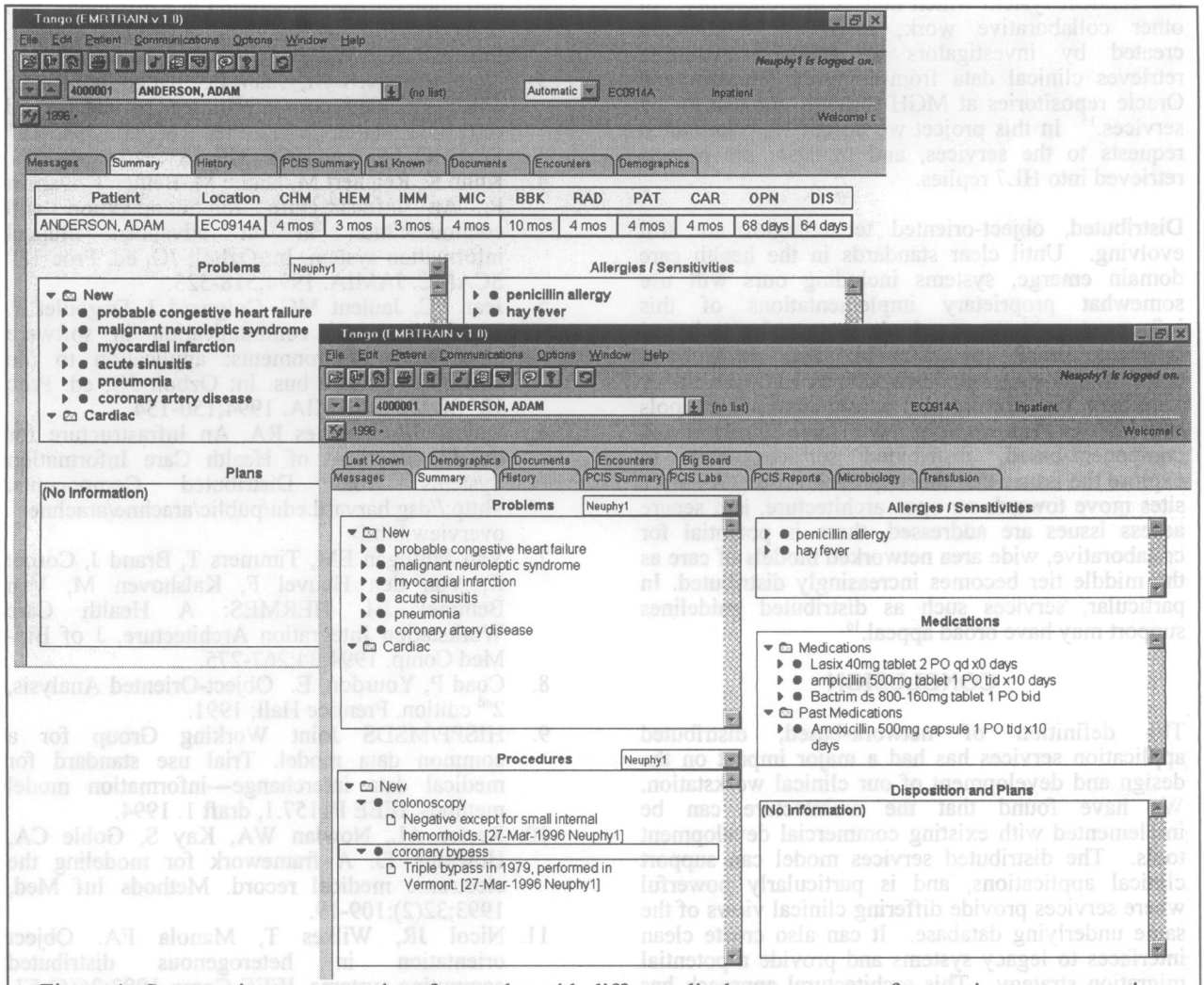

Figure 4. Same patient, two user interface styles with different display components for a patient summary view

DISCUSSION

Using a distributed services architecture has led to flexibility in application design because the client application is uncoupled from common application or "business" services. Services can be implemented once for use by many applications. Changes to clinical application logic (as opposed to interface behavior) can be changed once in a network service without needing to propagate a new application or services to all client workstations.

Network-based services can also be distributed over a many small, inexpensive servers, reducing potentially the overall cost of deploying applications. Moreover, a service-based architecture can be scaled up smoothly: we have added additional middle tier servers and replicated services onto these machines without having to decommission older servers. Client applications can be routed to the most available service on any middle tier server to perform load balancing. This architecture does depend on a robust network, which is a general requirement for all client-server architectures. Since there can be an increased number of points of failure with many application servers, requests for services are brokered intelligently, or "failed-over", to active servers when a targeted server fails.

A major advantage of the loose coupling between the client application and the services layer is the ability to utilize new application paradigms. Our component-based approach is one example, while another is the World Wide Web (WWW) and the Internet. Others have already shown that existing systems can be leveraged using WWW technology, where changes on network servers are available immediately to all client browsers.[14,15] In two different WWW projects, colleagues in our laboratory have been able to take advantage of the distributed middle tier services we have deployed. In one project, a WWW application uses the existing PatientIndexManager and TestManager services to provide a referring physician's interface into the MGH clinical repository. A simple TCP/IP sockets-based HyperText Transer Protocol (HTTP) adapter interfaces with the distributed object services. Application programmers needed no knowledge of

the Tandem system which houses the repository. In other collaborative work, a WWW application created by investigators at another institution retrieves clinical data from both the Tandem and Oracle repositories at MGH through our distributed services.[15] In this project we accept HL7 formatted requests to the services, and translate the objects retrieved into HL7 replies.

Distributed, object-oriented technologies are still evolving. Until clear standards in the health care domain emerge, systems including ours will use somewhat proprietary implementations of this infrastructure.[5,7] As standards mature for technical interfaces such as CORBA and OLE, and content/messaging standards such as HL7, the ability to utilize this architecture using commercial tools should expand rapidly. We have implemented component-based, distributed services now to explore the issues that this approach raises. As more sites move towards an open architecture, and secure access issues are addressed, there is potential for collaborative, wide area networked models of care as the middle tier becomes increasingly distributed. In particular, services such as distributed guidelines support may have broad appeal.[16]

CONCLUSION

The definition of network-based, distributed application services has had a major impact on the design and development of our clinical workstation. We have found that the architecture can be implemented with existing commercial development tools. The distributed services model can support clinical applications, and is particularly powerful where services provide differing clinical views of the same underlying database. It can also create clean interfaces to legacy systems and provide a potential migration strategy. This architectural approach has led us to define a number of core clinical services as well as adopt a specific data model for clinical objects. The flexibility of the network-based service layer has allowed others to explore quickly the development of other clinical applications using WWW and Internet development technology.

Acknowledgments

Special thanks to Fareeda Osman, Mary Morgan, Rick Cooper, Kevin Smith, Dan Salo, Margaret Moran, Vivian Gainer, and Yan Hoi Lee. This work was supported in part by grant LM05854 and training grant LM7092 from the National Library of Medicine, as well as grants from the Hewlett-Packard Corp.

References

1. Chueh HC, Barnett GO. Client-Server, Distributed Database Strategies in a Healthcare Record System for a Homeless Population. JAMIA. 1994;1: 186-198.
2. Lemaitre D, Sauquet D, Fofol I, Tanguy L, Jean FC, Degoulet P. Legacy systems: managing evolution through integration in a distributed and object-oriented environment. In Gardner RM, ed. Proc 19th SCAMC. JAMIA. 1995; 132-136.
3. Doré L, Lavril M, Jean FC, Degoulet P. An object oriented computer-based patient record reference model. In Gardner RM, ed. Proc 19th SCAMC. JAMIA. 1995; 377-381.
4. Kühn K, Reichert M, Nathe M, Beuter T, Dadam P. An infrastructure for cooperation and communication in an advanced clinical information system. In: Ozbolt JG, ed. Proc 18th SCAMC. JAMIA. 1994;518-523.
5. Jean FC, Jaulent MC, Coignard J, Degoulet P. Distribution and communication in software engineering environments: application to the HELIOS software bus. In: Ozbolt JG, ed. Proc 18th SCAMC. JAMIA. 1994;150-154.
6. Deibel SR, Greenes RA. An Infrastructure for the Development of Health Care Information Systems from Distributed Components. <http://dsg.harvard.edu/public/arachne/arachne_overview.html>
7. Van Mulligan EM, Timmers T, Brand J, Cornet R, Van den Heuvel F, Kalshoven M, Van Bemmel JH. HERMES: A Health Care Workstation Integration Architecture. J of Bio-Med Comp. 1994;34:267-275.
8. Coad P, Yourdon, E. Object-Oriented Analysis, 2nd edition. Prentice Hall; 1991.
9. HISPP/MSDS Joint Working Group for a common data model. Trial use standard for medical data interchange—information model methods. IEEE P1157.1, draft 1. 1994.
10. Rector AL, Nowlan WA, Kay S, Goble CA, Howkins TJ. A framework for modeling the electronic medical record. Methods Inf Med, 1993;32(2):109-19.
11. Nicol JR, Wilkes T, Manola FA. Object orientation in heterogenous distributed computing systems. IEEE Comp 1993;26(6):57-67.
12. Rocha RA, Huff SM, Haug PJ, Warner HR. Designing a controlled medical vocabulary server: the VOSER project. Comp Bio Res. 1994;27(6):472-507.
13. Forman BH, Cimino JJ, Johnson SB, et al. Applying a controlled medical terminology to a distributed, production clinical information system. In Gardner RM, ed. Proc 19th SCAMC. JAMIA. 1995; 421-425.
14. Cimino JJ, Socratous S, Clayton PD. Internet as clinical information system: application development using the World Wide Web. *JAMIA*, 1995:2(5):273-284.
15. Kohane IS, Greenspun P, Fackler J, Cimino C, Szolovits P. Building National Electronic Medical Record Systems via the World Wide Web. JAMIA. 1996;3(3):(in publication).
16. Barnes MR, Barnett GO. An architecture for a distributed guidelines server. In: Gardner RM, ed. Proc 19th SCAMC. JAMIA. 1995; 233-237.